Temporal Token Calculus: a Temporal Reasoning Approach for Knowledge-Based Systems*

Lluís Vila Gonzalo Escalada-Imaz

IIIA – CSIC
Camí de Sta. Bàrbara, s/n, 17300 Blanes, Catalonia, Spain
{vila,gonzalo}@ceab.es

Abstract

Most of AI research on temporal reasoning has been devoted to either exploring constraint-based temporal deduction techniques or investigating diverse logics extended with time. Nevertheless the formal study of deductive systems for such temporal extended logics has received little attention. This paper presents a general framework for temporal reasoning in knowledge-based systems resulting from embedding a temporal reasoner into a general calculus. From a representational point of view it is based on the notions of temporal token and temporal constraint. The logic is formally defined as a particular many-sorted predicate calculus and provided with an complete and sound inference system composed of non-temporal and temporal inference rules. Moreover, a deductive procedure is presented and analyzed. It is a general forward chaining algorithm for which soundness and completeness are guaranteed.

1 Introduction

Representation and reasoning about time is relevant to many AI areas like prediction, diagnosis, explanation, planning, scheduling, narrative understanding, software engineering, etc. Since the very beginning [11], AI research on temporal reasoning has been underlayed by the assumption that one can 1. define what time is made of and 2. develop specialized temporal deductive techniques for it, independently, up to some extent, of the way one constructs a temporal logic [1]. Therefore, we find, on the one hand, a large area of research devoted to develop efficient algorithms for the various

*Acknowledgments. We thank Carles Sierra and Jordi Levy for their helpful comments on earlier drafts. This work has been partially supported by Spanish MEC grant FPI-PN90 77909683 and by CICYT project *ARREL* (TIC92-0579-C02-01).

[1] In this spirit, Shoham[19] states an ontological distinction between the temporal entity over which one interprets assertions and the entity used as temporal primitive.

forms of purely temporal representations, either based on points or intervals, either made of qualitative, quantitative or both types of constraints, etc. And we find, on the other hand, a large set of proposals about the way time may be incorporated, eventually as an extension of a well-known logic formalism, to construct a temporal logic (see [?] for a comparative survey on both areas). Nevertheless the formal study of deductive systems for such temporal logics has received little attention. One must consider the logic resulting from embedding a temporal reasoner into a general calculus. The inference system will incorporate temporal specific inference rules and general ones. The deduction procedures will rely on specific algorithms and data structures specially designed for representing temporal assertions.

We aim at designing a temporal reasoning system suitable to Data- and Knowledge-based systems, specifically those related to prediction, monitoring and planning tasks. In this paper we do not only propose a particular temporal logic but we also furnish its inference system, we study its logical properties and we explore a forward computational procedure for it.

The main features of our temporal-extended language are the following: Ontologically it is based on the notion of temporal token. The temporal aspects are expressed through temporal predicates encoded by a specialized representation based on metric constraints. The sentences of the language are Horn-like clauses where *existential* quantifiers may appear in the head of the rule. We benefit from the temporal constraints formalism which is able to efficiently encode an infinite number of relationships into a finite representation.

In next section we discuss the main representational issues our approach is based on. In section 3 we formally present the *temporal token calculus* by giving its syntax, semantics and inference rules and showing soundness and completeness. We also provide a forward chaining deductive procedure and analyze its properties. In section 4 we discuss the relation of the temporal token calculus with previous work.

2 Representational Issues

2.1 Temporal Tokens

Reified temporal logic has been widely preferred as the logical form for temporal reasoning within the AI community [?]. Reification proposes according the status of terms in a first-order logic to expressions which we would normally regard as propositions. The increased expressive power provided by such a construction is useful for modeling some concepts specially relevant to problem solving like *causality* and *action*. Examples are [13, 2, 15, 8]. Recently, though, reified logics have come under attack. Main criticisms are (a) the inherent complexity of the reified construction which can be unnecessary [3, 9, 22], (b) the artificial and ad hoc nature of both (i) terms with propositional content and (ii) full ontological entities being their semantical referents [9] and (c) some expressiveness shortcomings [22].

The ontology of temporal reified approaches is based on the notion of *proposition type* [2] -for short we shall refer to it by *type*. A *type* is a generic atemporal atomic

[2]We shall distinguish between the two classical temporal entities *event* and *state* -the later also referred as property, fact, relationship or fluent in the literature.

proposition which, in some sense, denotes the set of all those particular instances of the proposition at different particular times. For instance, Allen[2] would express the sentence "Mary has been promoted from lecturer to professor on 1985" by forming the propositional term *promote*(*Mary, lecturer, professor*) which is a *type*, and using it as a parameter of the *Temporal Occurrence Predicate*(TOP) OCCURS to form the expression OCCURS(*promote*(*Mary, lecturer, professor*), 1985). This formula picks out the particular *instance* of a certain generic *type* -*promote*(*Mary,lecturer,professor*)- that *occurs* at a certain particular time. Such an implicit idea of instance is captured by the notion of *proposition token* -*token* for short. Tokens have been "explicitly" introduced in the ontology of a number of approaches in Temporal Reasoning. Dean and McDermott's TMM[5], Kowalski & Sergot's *Event Calculus*[12] (only for events), Galton's token-based *temporal arguments* approach [9], and the recent Vila & Reichgelt's *token reification*[22].

Since simplicity is of primary interest for our sort of applications, the approach we follow here is fully based on temporal tokens which are introduced as an additional argument (similarly to Event Calculus for events or Galton's unreification). Like Galton and Vila & Reichgelt we take a symmetric ontological definition based on tokens for both states and events. The language used here is a restricted instance of a many-sorted first-order language and reification is not used[3]. Tokens are introduced just as an additional sort. So far, we do not introduce any characterization of different *temporal entities* (events, states, processes ...) and therefore defining *temporal occurrence predicates* becomes unnecessary. Thus the notion of *token* is homogeneously used for expressing any temporal occurrence (both an events and states).

2.2 Describing Tokens by Semantic Case Decomposition

Information is usually incomplete in real domains. Event descriptions do not always include information related to every event feature. Such a situation holds as well for states. First-order logic based representations do not appear to be much appropriate to support such a description incompleteness. The features of events and states are just represented as different arguments of the same predicate. Since predicates arity is not variable, dealing with unknown features requires carrying on a considerable amount of existential quantifiers. The semantic cases knowledge representation paradigm proposes having a predicate related to each token feature. For example, instead of representing the previous example as *promote*(*Mary*, $sk f_i$, *professor*, 1985) we would write ...

$$act(tt_1, promote) \textbf{ and } \quad actor(tt_1, Mary) \textbf{ and}$$
$$destination(tt_1, professor) \textbf{ and } \quad \text{TIME}(tt_1, 1985)$$

Semantic case decomposition allows to represent just the pieces of information which are known and ignoring the rest, as Kowalski & Sergot propose[4]. We propose, here, using it not only for events but also for states description.

[3]We leave the issue of applying some sort of reification as a matter of future research in case it is judged to be of practical interest.

[4]Such a technique is already known in philosophy (D. Davidson [4]).

2.3 Using Metric Temporal Constraint Networks as the Basis for Token Management

Metric constraints are numeric constraints on the temporal distance between pairs of time points. Networks of metric temporal constraint have been proven to be an efficient formalism for encoding temporal knowledge and answering queries about it, though they may be less expressive than some interval-based approaches. We restrict here to the Simple Temporal Problem case [6] where a constraint is just formed by a pair of values denoting the lower and upper bound. Temporal constraint networks turn out to be a suitable formalism to express different notions concerning the tokens[5]:

Token. As we introduced above, the ontology of our approach is fully based on the notion of token. We simply need some functions to map each token to its relevant time points (TIME, BEGIN and END). A special instantaneous token tt_0 is introduced just to be the token occurring at the *initial time point* T_0 -thus $\text{TIME}(tt_0) = t_0$- that is used as a *common reference time point.*

Absolute and relative temporal relations. Absolute temporal information is expressed as temporal constraints between the initial time point and the time point at hand. For example, "Mary being hired as a lecturer on 1980" is expressed by $\text{DIST}(t_0, \text{BEGIN}(tt_1), 1980, 1980)$. Relative temporal information between two tokens is straightly expressed by temporal binary constraints between their endpoints.

Precise and vague temporal information. Temporal constraint networks provide a straight forward way to cope with vagueness of temporal information. Temporal constraints are a particular formalism for representing uncertainty on the temporal distance between two time points by declaring a range interval (lower bound, upper bound).

Metric and qualitative temporal relations. Qualitative is just a particular type of vagueness that involves $+\infty$, $-\infty$, $+\epsilon$ and/or $-\epsilon$ values. With these values, any of the 13 possible qualitative relations between two intervals [1] can be expressed as a set of temporal constraints. Furthermore, one can express the convex subset of the interval algebra though not the full Allen's interval algebra [23].

Token Inconsistency and Queries. Inconsistency of a set of tokens and queries about them can be easily checked/answered by applying constraint propagation techniques[6].

Token clipping. Many applications need to guarantee that two tokens do not overlap. For instance, it is common in planning where incompatible states for resources or different dependent tasks must fit into a single plan. In such cases one may require *token clipping* to accomplish such a purpose by imposing the adequate further constraints which eventually will stretch possible token durations.

[5] Tokens are notated by $tt_1, \ldots, tt_n$ and a constraint between the time points t, t' by $\text{DIST}(t, t', a, b)$

3 The Temporal Token Calculus

In this section we present our approach[6] for reasoning about events, states and time based on the representational issues discussed in previous section. Our ideas about (i) the ontological notion of token and (ii) giving a special status to time are reflected in a specific distinction between four different sorts: *tokens*, *values* -values for the attributes coming from the decomposition based on semantic cases-, *times* -time points related to token endpoints- and *durations* -distances between time points.

3.1 Syntax

We take a restricted many-sorted first-order language $\mathcal{L}_{\mathcal{TK}}$ with the following four sorts: *tokens*, *values*, *times* and *durations*. We have many-sorted signature $\langle S_{\mathcal{TK}}, \Sigma_{\mathcal{TK}} \rangle$, where $S_{\mathcal{TK}} = \{\mathcal{TOK}, \mathcal{VAL}, \mathcal{T}, \mathcal{DU}\}$ and $\Sigma_{\mathcal{TK}}$ is the union of the following indexed family of non-empty disjoint denumerable sets of symbols:

Constant symbol sets: $tt_1, \ldots, tt_i{:}\mathcal{TOK}$, $v_1, \ldots, v_i{:}\mathcal{VAL}$, $t_1, \ldots, t_i{:}\mathcal{T}$ and $d_1, \ldots, d_i{:}\mathcal{DU}$.

Function symbols: We have only the functions $\text{TIME}, \text{BEGIN}, \text{END} : \mathcal{TOK} \rightarrow \mathcal{T}$

Predicate symbols:

- *token-to-token predicate symbols* $\mathcal{P}_{\mathcal{TOK\_TOK}} : [\mathcal{TOK}]^n$, $n \geq 1$ (these are predicates like INITIATES, TERMINATES, CAUSES, INCOMPATIBLE, EXCLUSIVE, ...),
- *attribute predicate symbols* $\mathcal{P}_{\mathcal{ATT}} : \mathcal{TOK} \times \mathcal{VAL}$,
- *value predicate symbols* $\mathcal{P}_{\mathcal{VAL}} : \mathcal{VAL} \times \mathcal{VAL}$ (this set contains the value equality relation symbol $=_v$),
- a *temporal predicate symbol* $\text{DIST} : \mathcal{T} \times \mathcal{T} \times \mathcal{DU} \times \mathcal{DU}$,
- two *duration predicate symbols* $=, \preceq : \mathcal{DU} \times \mathcal{DU}$.

We have, also an indexed family of disjoint denumerable sets of variable symbols $V_{\mathcal{TK}} = \bigcup_{i \in S_{\mathcal{TK}}} V_i$, usually noted $V_{\mathcal{TOK}} = TT_1, \ldots, TT_i$, $V_{\mathcal{VAL}} = V_1, \ldots, V_i$, $V_{\mathcal{DU}} = DU_1, \ldots, DU_i$ and $V_{\mathcal{T}} = T_1, \ldots, T_i$.

The set of first-order terms $\mathcal{T}(\langle S_{\mathcal{TK}}, \Sigma_{\mathcal{TK}} \rangle, V_{\mathcal{TK}})$ is defined as usual.

Definition 1 *An* **atomic predicate** *or* **atom**, *for short, is of the form* $P(term_1, \ldots, term_n)$ *where* P *is an n-ary predicate symbol and* $term_i$ *are terms of the appropriate sort according to the type of the predicate. If* P *is a temporal predicate symbol then it is a* **temporal atom** *otherwise it is an* **atemporal atom.**

[6] We call it *temporal token calculus* since the token is our basic temporal entity for both events and states.

Definition 2 (well-formed formula -wff- or formula) *A well-formed formula or formula is of the form*

$$B_1 \textbf{ and } \ldots B_i \ldots \textbf{ and } B_m \Rightarrow H_1 \textbf{ and } \ldots H_j \ldots \textbf{ and } H_n$$

with $m \geq 0$ and $n \geq 0$, where B_i and H_j are atoms.

Like in *definite clauses*, the conjunction of B_i is called *body* or *antecedent* and the conjunction of H_j is called *head* or *conclusion*. We distinguish between three types of formula according to the form of the *body* and the *head*: **fact**, when $m = 0$ (empty body) and every H_j is a ground atom, **query**, when n=0 (empty head),**rule**, otherwise. A *knowledge base* is a couple $\mathcal{KB} = \langle \mathcal{FB}, \mathcal{RB} \rangle$ of a set of facts $\mathcal{FB}$ and a set of rules $\mathcal{RB}$.

All variables occurring in the body of a rule are considered as being universally quantified over the whole formula. All the variables occurring in the head of a rule and not occurring in its body are considered as being *existentially* quantified. In the promotion example, for instance, the fact that any promotion to professor is decided by the principal would be formalized as

$$\begin{array}{ll} act(TT, promote) \textbf{ and} & \\ destination(TT, professor) & \Rightarrow \quad decided(TT, principal) \end{array}$$

To express that any promotion event for a person to a certain rank "initiates" a state for this person having this rank we write

$$\begin{array}{lll} act(TT_1, promote) \textbf{ and} & \Rightarrow & act(TT_2, rank) \textbf{ and} \\ actor(TT_1, A) \textbf{ and} & & actor(TT_2, A) \textbf{ and} \\ destination(TT_1, X) & & rankvalue(TT_2, X) \textbf{ and} \\ & & \textsc{Initiates}(TT_1, TT_2) \end{array}$$

TT, TT_1, A and X are universally quantified but TT_2 is existential.

We claim that this definition covers all those syntactic forms which meaningfully "express" relations between sets of tokens fulfilling certain properties[7]. We deal with existentially quantified expressions by using standard skolemization.

3.2 Semantics

Definition 3 (Interpretation) *An interpretation $\Im$ is a tuple $\langle \mathcal{D}, \leq, \mathcal{M} \rangle$, where*

- *$\mathcal{D} = \mathcal{D}_{\mathcal{TOK}} \cup \mathcal{D}_{\mathcal{VAL}} \cup \mathcal{D}_{\mathcal{DU}}$ is a set of non-empty subdomains of tokens, values and durations, respectively.*
- *$\leq$ is a binary linear ordering relation on $\mathcal{D}_{\mathcal{DU}}$. According to this ordering,* addition *(+)*, subtraction *(-)*, maximum *(max)* and minimum *(min) operations are defined over $\mathcal{D}_{\mathcal{DU}}$ which includes d_0 that is the neutral element with respect to $+$ and $-$ operations.*

[7]This definition excludes forms which either (i) have an existentially quantified body or (ii) have a universal quantification over a variable appearing in the head but not in the body. Such forms are meaningless in the context of stating relations between sets or classes of tokens.

- *$\mathcal{M}$ is a* meaning *function which associates:*
  - *each constant symbol with a domain element of its sort $\mathcal{M} : \Sigma_{\mathcal{TK}} \longrightarrow \mathcal{D}$.*
  - *each function symbol with a function $\mathcal{D}_{\mathcal{TOK}} \rightarrow \mathcal{D}_{\mathcal{DU}}$.*
  - *each neither temporal nor duration n-ary predicate symbol with a subset of the cartesian product over the subdomains determined by its signature.*

Definition 4 (matcher) *A matcher θ for a formula F is a substitution of any variable occurring in F by a ground term of the same sort. Let $\theta.F$ denote the application of the matcher over F. A matcher θ* agrees *with a matcher θ' if any variable in the domain of both is substituted by the same ground term.*

Definition 5 ($\models$) *The interpretation $\Im$ satisfies a formula F, noted $\Im \models F$, under the following inductively defined conditions:*

1. *$\Im \models B \Rightarrow H$* iff *whenever $\Im \models \theta.B$ for a certain matcher θ, there exists at least one matcher θ' of H that* agrees *with θ such that $\Im \models \theta'.H$.*

   In the following items, let A_i, $term_i$, t_term_i, d_i be ground atoms, ground terms, ground time terms and ground duration terms respectively.

2. *Let $P(term_1, \ldots, term_n)$ be a* ground atemporal atom, *$\Im \models P(term_1, \ldots, term_n)$* iff
   $$\langle \mathcal{M}(term_1), \ldots, \mathcal{M}(term_n) \rangle \in \mathcal{M}(P)$$

3. *$\Im \models \textsc{Dist}(t\_term_1, t\_term_2, d_1, d_2)$* iff
   $$\mathcal{M}(d_1) \leq \mathcal{M}(t\_term_2) - \mathcal{M}(t\_term_1) \leq \mathcal{M}(d_2)$$

4. *$\Im \models d_1 = d_2$* iff *$\mathcal{M}(d_1)$ is equal to $\mathcal{M}(d_2)$*

5. *$\Im \models d_1 \preceq d_2$* iff *$\mathcal{M}(d_1) \leq \mathcal{M}(d_2)$*

6. *$\Im \models A_1$* **and** *$\ldots A_i \ldots$* **and** *A_n* iff *$\Im \models A_i$ for every i.*

Definition 6 *As usual a wff* F *is* satisfiable *if there is an interpretation $\Im$ such that $\Im \models$ F. A wff* F *is* valid *if $\Im \models$ F for every interpretation $\Im$. A set of wffs $\mathcal{KB}$* logically entails *a wff F (noted $\mathcal{KB} \models F$) if for every interpretation $\Im$ such that $\Im \models \mathcal{KB}$ then $\Im \models F$.*

3.3 Inference

A main issue of this work is providing our temporal logic with a deductive system. We begin by giving a set of inference rules. We distinguish between two types of inference rules: (1) those that are applied to general formula and (2) those that specifically are applied over temporal atoms.

Let t_i be temporal terms and $l_{ij}, u_{i'j'}, d_k$ be duration terms for any i, j, k.

$$\frac{}{\text{DIST}(t_1,t_1,0,0)}\{\text{Reflexivity}\} \qquad \frac{\text{DIST}(t_1,t_2,l_{12},u_{12})}{\text{DIST}(t_2,t_1,-u_{12},-l_{12})}\{\text{Symmetry}\}$$

$$\frac{\text{DIST}(t_1,t_2,l_{12},u_{12}),\text{DIST}(t_2,t_3,l_{23},u_{23})}{\text{DIST}(t_1,t_3,l_{12}+l_{23},u_{12}+u_{23})}\{\text{Trans.}\} \qquad \frac{\text{DIST}(t_1,t_2,l_{12},u_{12}),\text{DIST}(t_1,t_2,l'_{12},u'_{12})}{\text{DIST}(t_1,t_2,max(l_{12},l'_{12}),min(u_{12},u'_{12})}\{\text{Intersection}\}$$

$$\frac{\text{DIST}(t_1,t_2,l_{12},u_{12}),d_1 \preceq l_{12},u_{12} \preceq d_2}{\text{DIST}(t_1,t_2,d_1,d_2)}\{\text{Inclusion}\}$$

General inference is performed through a single inference rule which is a sort of *modus ponens* (MP):

$$\frac{B_1 \text{ and } \ldots \text{ and } B_m \Rightarrow H_1 \text{ and } \ldots \text{ and } H_n, B'_1, \ldots, B'_m,\ \exists\theta.\ \forall i.\ \theta.B_i = B'_i}{\theta.H_1, \ldots, \theta.H_n}$$

Moreover, we have a simple axiom to describe syntactical equality between values: $V : \mathcal{VAL} =_v V : \mathcal{VAL}$. We have, also, the axioms for $\preceq$ to be a *linear order* over $\mathcal{D}_{\mathcal{DU}}$ terms. We finally need an axiom ensuring the existence of the coarsest constraint between any two pair of time points, $\text{DIST}(T, T', -\infty, +\infty)$

3.4 Soundness and Completeness

Theorem 1 (soundness) *For any knowledge base $\mathcal{KB}$ and atomic formula P: If $\mathcal{KB} \vdash P$ then $\mathcal{KB} \models P$.*

Let us now state a completeness result. We shall assume consistency for the knowledge base at hand.

Let's first introduce some notational conventions. We note facts in a $\mathcal{KB}$ by $f_1, \ldots, f_i$ and rules by $r_1, \ldots, r_i$. $Ant(r)$ denotes the body of the rule r and $Concl(r)$ its head.

The proof of completeness employs the following definitions. We view a knowledge base as the set of all the possible instantiations:

Definition 7 (ground KB) *Let $\theta_1, \ldots, \theta_i, \ldots$ be all the possible different matchers for the formula $f_1, \ldots, f_k, r_1, \ldots, r_n$ of a given knowledge base $\mathcal{KB}$. The set of* ground knowledge bases, *noted $Gd(\mathcal{KB})$, is the set $\theta_1.\mathcal{KB} \cup \ldots \cup \theta_i.\mathcal{KB} \cup \ldots$ where $\theta_i.\mathcal{KB} = \{f_1, \ldots, f_k, \theta_i.r_1, \ldots, \theta_i.r_n\}$.*

We consider all the possible paths we can build between a pair of time points by chaining temporal predicates.

Definition 8 (D) *Let $D(t,t',a,b)$ denote a set of* DIST *predicates included in a set S of literals such that $D(t,t',a,b) = \{\text{DIST}(t_i,t_{i+1},a_i,b_i) \mid 1 \leq i < n, t_1 = t, t_n = t', \text{DIST}(t_i,t_{i+1},a_i,b_i) \in S \vee \text{DIST}(t_{i+1},t_i,-b_i,-a_i) \in S, a = \sum_{i=1}^{n-1} a_i, b = \sum_{i=1}^{n-1} b_i\}$.*

And we consider also the most constrained path induced by the set of paths between any given pair of time points.

Definition 9 *($\mathcal{D}$) Given a set of literals S, consider all the paths $D_1(t,t',a_1,b_1)$ S, ..., $D_k(t,t',a_k,b_k) \subseteq S$ between a pair of time points t,t'. Then $\mathcal{D}(t,t',\alpha,\beta)$ $\bigcup_{i=1}^{k} D_i(t,t',a_i,b_i)$, where $\alpha = max_i\{a_i\}$ and $\beta = min_i\{b_i\}$.*

Finally let's define a certain deductive closure for a $\mathcal{KB}$.

Definition 10 *Let $\mathcal{S} = \bigcup_{j=0}^{\infty} S_j$, where S_j are built from $Gd(\mathcal{KB})$ as follows:*

1. $S_0 = \{f_1, \ldots, f_n\}$
2. *For each r such that $r \in Gd(\mathcal{KB})$, if $\forall P \in \text{Ant}(r)$*
   - *(a) if $P \neq \text{DIST}$ then $P \in \bigcup_{j \leq k} S_j$*
   - *(b) if $P = \text{DIST}(t,t',a,b)$ then $\exists \mathcal{D}(t,t',\alpha,\beta) \subseteq \bigcup_{j \leq k} S_j$ with $a \leq \alpha$ and $\beta \leq b$*

   then $\text{Concl}(r) \in S_{k+1}$.

To prove the theorem we need the following propositions and lemmas.

Proposition 1 *$\mathcal{KB} \equiv Gd(\mathcal{KB})$.*

Proposition 2 *$\mathcal{KB} \models P$ iff $Gd(\mathcal{KB}) \models P$.*

Proposition 3 *$\mathcal{D}(t,t',\alpha,\beta) \vdash \text{DIST}(t,t',a,b)$ with $a \leq \alpha \leq \beta \leq b$.*

Proposition 4 *Assume that $\alpha \leq \beta$ for any $\mathcal{D}(t_i,t_j,\alpha,\beta) \subseteq \mathcal{S}$. Then, there exist an interpretation $\Im = \langle \mathcal{D}, \leq, \mathcal{M} \rangle$ that satisfies any predicate in $\mathcal{D}(t,t',\alpha,\beta)$, $\alpha \leq \beta$ i $\mathcal{M}(t') - \mathcal{M}(t) \in [\mathcal{M}(\alpha), \mathcal{M}(\beta)]$.*

Lemma 1 *$\mathcal{KB}$ is consistent iff $\forall t,t'.\ \mathcal{D}(t,t',\alpha,\beta) \rightarrow \alpha \leq \beta$.*

Lemma 2 *If $P \in \mathcal{S}$ then $\mathcal{KB} \vdash P$.*

Lemma 3 *$\mathcal{KB} \models P$ and $P \neq \text{DIST}$ and $\mathcal{KB}$ is consistent then $P \in \mathcal{S}$.*

Lemma 4 *$\mathcal{KB} \models P$ and $P = \text{DIST}(t,t',a,b)$ and $\mathcal{KB}$ is consistent ther $\exists D(t,t',\alpha,\beta) \subseteq \mathcal{S}$ with $a \leq \alpha \leq \beta \leq b$.*

Theorem 2 *For any* consistent *knowledge base $\mathcal{KB}$ and atomic formula P: If $\mathcal{KB} \models P$ then $\mathcal{KB} \vdash P$.*

3.5 Forward Deductive System

In this section we present a simple deductive algorithm working in a bottom-up fashion. *Forward* proceeds by processing new facts –those generated during last iteration– during each iteration. Facts are deduced by application of our *Modus Ponens* inference rule over each possible rule in the $\mathcal{KB}$. The matchers are always constructed by using at least one new fact. Notice that temporal and non-temporal facts receive different treatments: a special function Add_facts_T is used to add new temporal facts to the KB (Add_facts_T is discussed in section 3.6).

function *Forward* (FB_T,FB_{NT},RB) **is**
{FB_T,FB_{NT} are temporal and nontemporal fact bases; RB is a rule base}
1. new_fact$_T$ $\leftarrow$ FB_T; new_fact$_{NT}$ $\leftarrow$ FB_{NT}; $FB'_T \leftarrow \emptyset$; $FB'_{NT} \leftarrow \emptyset$
2. **while** new_fact$_T$ $\neq \emptyset \vee$ new_fact$_{NT} \neq \emptyset$ **do**
3. **if** new_fact$_T \neq \emptyset$ **then** $FB'_T \leftarrow$ *Add_facts*$_T(FB'_T$, new_fact$_T)$; new_fact$_T \leftarrow \emptyset$ **endif**
4. **if** new_fact$_{NT} \neq \emptyset$ **then** $FB'_{NT} \leftarrow FB'_{NT} \cup$ new_fact$_{NT}$; new_fact$_{NT} \leftarrow \emptyset$ **endif**
5. **foreach** r $\in RB$ **do**
6. **foreach** $\theta \in$ *Match*$(FB'_T, FB'_{NT}$, *Ant*$(r))$ **do**
7. ⟨deduced_facts⟩ $\leftarrow$ *New_deduction*$(\theta$.*Concl*$(r))$;
8. new_fact$_T \leftarrow$ new_fact$_T \cup$ deduced_facts$_T$;
9. new_fact$_{NT} \leftarrow$ new_fact$_{NT} \cup$ deduced_facts$_{NT}$
endforeach
endforeach
endwhile
10. **return**$(FB'_T \cup FB'_{NT})$
endfunction

Match proceeds iteratively over the body of a rule to construct the set of matchers that identify a subset of the fact base with such a body. For each conjunctive atom, in accordance with its predicate symbol, *Match* retrieves either temporal or non temporal facts that "match" with the atom. These are *combined* with the partial matchers constructed up to that point applying the cartesian product –the classical combination of substitutions. To obtain the set of matchers between a non-temporal atom and the base FB_{NT}, a classical first-order matcher is used scanning all the facts in this base (*Retrieve_facts*$_{NT}$). In case the atom is temporal then *Retrieve_facts*$_T$ function is used (discussed in section 3.6).

function *Match* (FB_T,FB_{NT},B) **is**
{FB_T,FB_{NT} are temporal and nontemporal fact bases and B a rule body}
$\Theta \leftarrow \{\epsilon\}$
while $\Theta \neq \{\}$ **and** $B \neq \emptyset$ **do**
$A \leftarrow$ *First*(B); $B \leftarrow B - A$;
if *Is_temporal*(A) **then** $\Theta_A \leftarrow$ *Retrieve_facts*$_T(FB_T, A)$ **else** $\Theta_A \leftarrow$ *Retrieve_facts*$_{NT}(FB_{NT}, A)$ **endif**
if $\Theta_A \neq \emptyset$ **then** $\Theta \leftarrow \{$*Combine*$(\theta, \theta') | \theta \in \Theta, \theta' \in \Theta_A)\}$ **endif**
endwhile
return(Θ)
endfunction

Soundness and Completeness. We want to show that the results returned by *Forward* are sound and correspond to all those facts that can be deduced from a given knowledge base. Assume that *Forward*(FB_T, FB_{NT}, RB) ends returning a set $FB'_T \cup FB'_{NT}$. Then we shall show that $FB_T \cup FB_{NT} \cup RB \vdash P \neq$ DIST (resp. $=$ DIST) if and only if $P \in FB'_{NT}$ (resp. *Retrieve_facts*$_T(FB_T, P)$ returns **Yes**).

We use several lemmas to show each property:

Lemma 5 *Let us suppose that* Retrieve_facts$_T(FB_T, P)$ *and* Retrieve_facts$_{NT}(FB_{NT}, P)$ *return a set of unifiers* Θ. *Then:*

- $P \neq$ DIST: $\theta \in \Theta$ *iff* $\exists f \in FB.\ f = \theta.P$.
- $P =$ DIST: $\theta \in \Theta$ *iff* $\theta.P =$ DIST(t, t', α, β) *and it holds that* $\mathcal{D}(t, t', \alpha, \beta)$ *according to definition 9 upon* FB_T.

Lemma 6 *Let us suppose that* Match$(FB_T, FB_{NT}$, Ant$(r))$ *returns a set of unifiers* Θ. *Then* $\forall B \in$ Ant(r). $\exists f \in FB_T \cup FB_{NT}$ *such that* $\theta.B = f$ *iff* $\theta \in \Theta$.

Theorem 3 (soundness) *Assume that* Forward(FB_T, FB_{NT}, RB) *ends returning a set* $FB'_T \cup FB'_{NT}$. *If* $P \neq$ DIST *and* $P \in FB'_{NT}$ *(resp.* $P =$ DIST *and* Retrieve_facts$_T(FB'_T, P)$ *returns* **Yes***) then* $FB_T \cup FB_{NT} \cup RB \vdash P$.

Proof: It is a direct consequence of the **if** sense of lemma 6 and the operations of the lines 7, 8, 9, 3 and 4 in *Forward* which guarantee the passing of temporal (resp. no-temp) facts from the conclusion of an applied rule to FB_T (resp. FB_{NT}). □

Lemma 7 *Let* $\varphi(P)$ *be the set of predicates in the deduction tree of* P. *If* $FB_T, FB_{NT}, RB \vdash P$ *then* Forward(FB_T, FB_{NT}, RB) *deduces each predicate in* $\varphi(P)$, *i.e.* $\varphi(P) \subseteq FB'_T \cup FB'_{NT}$.

Proof: This follows from the continuous application of every rule over each updated set $FB_T \cup FB_{NT}$ of a main iteration in *Forward*. □

Theorem 4 (completeness) *Assume that* Forward(FB_T, FB_{NT}, RB) *ends returning a set* $FB'_T \cup FB'_{NT}$. *If* $FB_T \cup FB_{NT} \cup RB \vdash P$ *and* $P \neq$ DIST *(resp.* $P =$ DIST*) then* $P \in FB'_{NT}$ *(resp.* Retrieve_facts$_T(FB_T, P)$ *returns* **Yes***).*

Termination of our deductive algorithm is not guaranteed. Since 1. recursive definition of predicates is allowed, and 2. the Herbrand Universe is potentially infinite due to the skolem functions introduction to deal with existential quantifiers, *Forward* may not terminate. We are indeed is case very similar to standard predicate calculus. Since our procedure is breath-first like, any fact which actually follows from $\mathcal{KB}$ will be derived after a certain computation time.

Theorem 5 (complexity). *The number of operations* Combine(θ, Θ) *can be exponential in a worst case.*

Proof: The proof is straight forward considering the fact that the number of ground terms of a rule head is given by the compatible substitutions obtained from the cartesian product of the set of substitutions for each predicate in the rule body. □

We should note that though the combination method turns out to be exponential in a worst case is not a categorical reason for disapproval regarding applications. There are several arguments supporting the interest for our proposal:

- We are approaching a logic that provides a remarkable expressiveness which allow us to suitably represent knowledge pieces and their temporal aspects. The worst-case complexity comes inherently from such an expressive power. The discouraging general properties of such highly expressive languages do not necessarily undermine their usefulness applicability. One may "usually" get reasonable running times for realistic applications [8].
- Some optimizations can be proposed to restrict exponentiality (some of them are discussed below).

3.6 Temporal Inference

A major feature in our temporal reasoning system is the representation of "pure" temporal knowledge by temporal constraints and its processing through a specialized device. As we have seen, the properties of the whole system hang on the characteristics of this temporal component. From a system architecture point of view one can imagine the temporal module as a "black box" which interfaces with the rest of the system through two functions: Add_facts_T and $Retrieve\_facts_T$. Functionally they take as input a temporal constraint and a temporal constraint network and give back different results. Add_facts_T returns a revised temporal constraint network [9]. $Retrieve\_facts_T$ returns either **false** in case the input constraint is inconsistent with the network or the set of most restricted (according to the network) feasible instantiations of the given constraint.

In this section we discuss the particular type of temporal constraints we take, the existing algorithms for dealing with them and the general properties a temporal module must exhibit. The DIST predicate has been originally introduced by Dean & McDermott in their TMM [5]. Dechter *et al.* [6] formulate networks of temporal constraints as a CSP and distinguish a simple case they name STP (*Simple Temporal Problem*) which corresponds to the case where one has only unary and binary constraints and they specify just a single metric interval of possible values. A set of temporal constraints expressed through our DIST predicate is a STP.

Diverse alternatives can be considered as algorithms for such interface functions in accordance with the representation of our constraint networks. Different representations yield different results on the trade-off between *addition* and *retrieval* procedures complexity. Dechter *et al.*'s completely connected graph representation (*d-graph*) allows for applying rather classical techniques for addition (the addition of a further

[8] PROLOG is an example: it is being fairly successfully used despite the fact that even *termination* is not guaranteed.

[9] We assume that the additional constraints are always consistent with the current network.

constraint and re-building of the d-graph is at worst $O(n^2)$ although it seems easily improvable [20]) and provides a constant time retrieval. Representations based on not completely connected graphs [17, 14, 10] provide lower addition cost though they pay some performance penalty at retrieval time. See the recent paper [24] for experimental results on the suitability of the various representations according to the characteristics of the application.

In general, the many alternatives for the temporal inference module must fulfill a basic condition independently of the approach they follow. Given a temporal constraint $tc = \text{DIST}(t_1, t_2, d_1, d_2)$ and a temporal constraint network *tcn*:

- If tc is ground **then** $Retrieve\_facts_T(tc, Add\_facts_T(tc, tcn)) =$ **true**.
- If tc is not ground **then** $Retrieve\_facts_T(tc, tcn) = \text{DIST}(t, t', \alpha, \beta)$ such that it holds $\mathcal{D}(t, t', \alpha, \beta)$ according to definition 9 upon the set of those facts that have been previously added to *tcn*.

4 Related Work

From a knowledge representational point of view this is an alternative approach to reified temporal logics. As discussed above we share with the Event Calculus the idea of introducing *temporal tokens* as a new sort as well as the classical knowledge representation technique of semantic case decomposition. Kowalski & Sergot apply them only for event descriptions. We use them for describing any temporal occurrence.

This work has many similarities with Dean & McDermott's TMM. From a logical definition point of view we supply our logic with formal definitions of its syntax, semantics [10] and an inference system. Such a system integrates the well-known *Modus Ponens* rule and a set of inference rules governing the behavior of the DIST predicate. They characterize the unary and binary temporal constraints of the STP case defined by Dechter *et al.*[6]. Our contribution of soundness and completeness proofs hangs on it. We do not know about similar results for other temporal reasoning systems. We claim that these proofs can easily be adapted for other types of constraints for which a sound and complete set of inference rules is known.

From a reasoning point of view, we developed a forward deduction algorithm which hangs on the functions Add_facts_T and $Retrieve\_facts_T$. As long as we have constraints of the STP type we can use the algorithms for adding a further constraint in a d-graph [20] and the definitions of feasible distances of feasible values provided by Dechter *et al.*. Alternatively we could use algorithms based on not fully connected graphs which for large constraint bases provide experimental efficiency gains [24]. The neat decoupling of the temporal reasoning component makes easy replacing simple binary constraints by more sophisticated types when required from application domain. One would introduce a different definition of temporal constraint in the language and would incorporate specialized algorithms through the Add_facts_T and $Retrieve\_facts_T$ functions. In particular we could introduce a sort of *disjunctive constraints* which would supplant the

[10] Shoham [19] remarks the lack of such definitions for the classical most relevant approaches to temporal reasoning systems and provides reasons for the convenience of doing so.

lack of the disjunctive connective in our logic. We are exploring the introduction of techniques for dealing with such constraints which is currently being a matter of study for the temporal reasoning research community[16, 18, 7].

5 Conclusions and Future Work

This paper defines *Temporal Token Calculus*, a general framework for temporal reasoning in knowledge-based systems. It is based on the notions of temporal tokens and encoding temporal aspects through temporal constraints. This logic is formally defined as particular many-sorted language of definite clauses. The *Simple Temporal Problem*[6] is taken as constraints language. An inference system is furnished where inference rules are distinguished between non-temporal and temporal ones and its soundness and completeness are proven. Moreover, we explore a deductive procedure. In this paper we present a general forward chaining algorithm for which termination, soundness and completeness are guaranteed. Though the worst case complexity of the algorithm is exponential due to the inherent complexity of the problem, we outline a number of optimizations.

A prototype of this system is currently being developed at the IIIA. It is implemented in MacIntosh Common Lisp and integrated in MILORD-II *shell*. We are working in a monitoring application (a farm management assistant) to show the suitability of the temporal token approach and the forward inference system presented here.

This work requires further investigation in two different lines:

Inference. The above outlined optimizations are being detailedly studied. Results could be relevant not only to our temporal logic but to theorem proving technology in general. There is further work on developing, similarly to the forward case, a backward and an incremental deduction algorithms. Probably it will motivate further investigation about inference control strategies which skillfully make the right decisions concerning the use of either the temporal or the general component of the inference system in order to gain efficiency. Possibly it will depend on the structure of the domain knowledge.

Expressiveness. Our logic is based on a restricted predicate calculus and a simple case temporal constraints which can be not expressive enough for certain applications. The underlying thesis of our approach is that we shall be able to circumscript the extensions needed to achieve higher expressive power to the temporal module.

References

[1] J. F. ALLEN. Maintaining knowledge about temporal intervals. Technical Report TR86, Department of Computer Science University of Rochester, 1983.

[2] J. F. ALLEN. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[3] F. BACCHUS, J. TENENBERG, and J. A. KOOMEN. A non-reified temporal logic. *Artificial Intelligence*, 52:87–108, 1991.

[4] D. DAVIDSON. The logical form of action sentences. In N. Rescher, editor, *The Logic of Decission and Action*. University of Pittsburgh Press, 1967.

[5] T. L. DEAN and D. V. McDERMOTT. Temporal data base management. *Artificial Intelligence*, 32:1987, 1987.

[6] R. DECHTER, I. MEIRI, and J. PEARL. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[7] R. DECHTER and E. SCHWALB. Compiling relational data into disjunctive structure: Empirical evaluation. In *AI/GI/VI'94*, 1994.

[8] A. GALTON. A critical examination of Allen's theory of action and time. *Artificial Intelligence*, 42:159–188, 1990.

[9] A. GALTON. Reified temporal theories and how to unreify them. In *IJCAI'91*, pages 1177–1182, 1991.

[10] A. GEREVINI, L. SCHUBERT, and S. SCHAEFFER. Temporal reasoning in timegraph I-II. *SIGART bulletin*, 4(3):T1–T4, 1993.

[11] K. KAHN and G. GORRY. Mechanizing temporal knowledge. *Artificial Intelligence*, 9:87–108, 1977.

[12] R. KOWALSKI and M. SERGOT. A logic-based calculus of events. *New Generation Computing*, 3, 1986.

[13] D. McDERMOTT. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6:101–155, 1982.

[14] S. A. MILLER and L. K. SCHUBERT. Time revisited. *Computational Intelligence*, 6:108–118, 1990.

[15] H. REICHGELT. A comparison of first-order and modal logics of time. In P. Jackson, H. Reichgelt, and F. van Harmelen, editors, *In Logic-based Knowledge Representation*, pages 143–176. The MIT press, 1989.

[16] R. SCHRAG, M. BODDY, and J. CARCIOFINI. Managing disjunction for practical temporal reasoning. In *KR'92*, pages 36–46, 1992.

[17] R. SCHRAG, J. CARCIOFINI, and M. BODDY. β-tmm manual (version b19). Technical Report CS-R92-012, Honeywell SRC, 1991.

[18] E. SCHWALB and R. DECHTER. Coping with disjunctions in temporal constraint satisfaction problems. In *AAAI'93*, pages 127–132, 1993.

[19] Y. SHOHAM. Temporal logics in AI: Semantical and ontological considerations. *Artificial Intelligence*, 33:89–104, 1987.

[20] L. VILA. Constraints on distances between temporal distances. Report de Recerca forthcoming, IIIA, 1993.

[21] L. VILA. A survey on temporal reasoning in artificial intelligence (revised version 93). Report de Recerca 93/19, IIIA, 1993.

[22] L. VILA and H. REICHGELT. The token reification approach to temporal reasoning. Technical Report 93/1, Dept. of Computer Science, UWI, 1993.

[23] M. VILAIN and H. KAUTZ. Constraint propagation algorithms for temporal reasoning. In *AAAI'86*, pages 377–382, 1986.

[24] E. YAMPRATOOM and J. ALLEN. Performance of temporal reasoning systems. *SIGART*, 4(3):T1–T4, 1993.

Extending Explanation–Based Generalization with Metalogic Programming

Stefano Bertarello
berta@morgana.usr.dsi.unimi.it
tel. ++39–2–55006327

Stefania Costantini
costanti@imiucca.csi.unimi.it
tel. ++39–2–55006259

Gaetano Aurelio Lanzarone
lanzarone@hermes.mc.dsi.unimi.it
tel. ++39–2–55006324

Dipartimento di Scienze dell'Informazione, Universita' degli Studi di Milano
via Comelico 39, 20135 Milano, ITALY

Abstract

In this paper we discuss Explanation–Based Generalization (EBG) in the context of metalogic programming. EBG is a technique for learning by examples that creates generalizations of given instances on the basis of background knowledge of the domain. The process can leave the deductive closure of the domain theory unchanged (theory reformulation) or even add new knowledge (theory revision). While Logic Programming has proved suitable for theory reformulation, less has been done for theory revision. In this paper we characterize the distinction between theory reformulation and revision by stating formal definitions, and propose a method for theory revision based on abstraction and on the treatment of exceptions.
We choose metalogic programming, and in particular the language Reflective Prolog, as the language for performing EBG, since the EBG process is metatheoretical in nature and since, for theory revision, the possibility of making statements about predicates is needed.

1. Introduction

Explanation–Based Learning (EBL) is a technique, developed in the field of machine learning, by which an intelligent agent can learn by examples [El89]. Differently from other methods of concept learning, EBL creates justified generalizations from training instances by relying on background knowledge of the domain. To emphasize that learning may involve generalizing the example(s), the term Explanation–Based Generalization (EBG) is commonly used. We will adopt the EBG term and attitude in the following since, especially in the context of logic programming, generalization is a natural outcome of the inference process.

Adopting Dietterich's definition (reported in [El89]), "a system is said to perform

knowledge–level learning only when there is a change in the 'deductive closure' of its domain theory". Standard EBG algorithms (thus most proposed EBG systems) do not change the deductive closure of the domain theory and therefore do not perform knowledge–level learning, but only a *theory reformulation*. By only changing the way concepts are expressed, theory reformulation is mainly oriented to make easier the task of recognizing further instances of the concept. On the contrary, enhanced EBG algorithms aim at building bridges over knowledge holes, and at making plausible leaps in the reasoning; in other terms, at performing knowledge–level learning, or *theory revision*.
Logic programming has proved to be a suitable context for programming EBG as theory reformulation. In [KM87], a metainterpreter is presented which performs generalization as a byproduct of standard SLD–derivation. In [VB88] the similarity is shown between this kind of generalization process and another well–known technique of functional and logic programming, namely partial evaluation.
Little has been done instead with respect to EBG in the sense of theory revision. In the EBL literature, first steps in this direction have been presented in [DK90].
We discuss the above–mentioned topics in the context of metalogic programming.

Our contribution is twofold. First, we give an original treatment to two open problems of the EBG theory: abstraction and exceptions. Second, we characterise the distinction between theory reformulation and theory revision by stating formal definitions and results.

With respect to abstraction, we propose a method able to fill in knowledge gap by exploiting taxonomic information. The method is domain–independent, where however the abstraction process must be guided by domain–dependent directives, expressed by the user in a predefined format. Exceptions (or, more generally, constraints) can also be defined, in order to ensure coherence of the new knowledge generated by abstraction.
We use metalogic programming, and precisely the language Reflective Prolog, for the following reasons. First EBG is in itself a metaprogramming task. Then, directives and constraints for theory revision are in general predications about predicates; since one of the objectives of this paper is a formal definition of EBG, the predications must be expressed at the metalevel in the context of a suitable formalism.

The paper is organized as follows. In section 2 we introduce EBG as it has been considered in literature, especially in the context of logic programming. In section 3 we face the problem of semantically characterising what is generated by the standard EBG process (theory reformulation). We show, under suitable definitions, that the program obtained is not strongly equivalent to the original one (equivalence under subsumption) but is weekly equivalent to it (equality of their least Herbrand models). In section 4 we consider extensions of the standard EBG process (theory revision), aimed to fill in knowledge gaps. The extensions are capable to treat heuristics represented by abstraction hierarchies with constraints and exceptions. We also consider that, though

the extended theory is clearly no longer equivalent to the original one, a characterisation of the consequences of the extended theory can still be given.

2. Defining EBG

In the following, we will take [KM87] as base reference w.r.t. EBG.

Explanation–Based Learning (EBL) is a technique by which an intelligent agent can learn through the observation of examples.

EBG is usually phrased in the literature as follows. Given in input:

(1) a Target Concept (TC): a predicate representing the concept to be learned/generalized;
(2) a Training Example (TE): a set of clauses (usually facts) constituting an example (i.e. a specific instance) of the target concept;
(3) a Domain Theory (DT): a set of facts and rules representing background knowledge about the domain, that are used in explaining why the training example is an instance of the target concept;
(4) an Operationality Criterion (OC): a set of predicates in terms of which TC has to be defined

EBG gives in output a definition of TC which generalizes TE and satisfies OC.

The EBG algorithm consists of two stages:
(1) construct an explanation in terms of the domain theory that shows how the training example satisfies the target concept definition.
Each branch of the explanation structure must terminate with an expression that satisfies the operationality criterion.
(2) determine a set of sufficient conditions under which the explanation holds, stated in terms that satisfy the operationality criterion.

In logic programming, the first step consists in proving that the TE is indeed an instance of TC, in such a way that all the leaves of the proof tree are operational predicates, while the second step consists in building a more general version of this proof that does not depend on any of the irrelevant (non–operational) features of TE.

An EBG algorithm as a Prolog metainterpreter is given in [KM87]. The following is a similar algorithm written in Reflective Prolog, which is an extension of Prolog amalgamating language and metalanguage to treat metalevel knowledge and metalevel reasoning (fully defined in [CL93]). A very short introdution to Reflective Prolog is reported in appendix B.

The reader wishing to compare this Reflective Prolog version of EBG with the Prolog one given in [KM87] will notice that there is not much difference; the metalevel features of Reflective Prolog will be more usefully exploited in the extensions discussed in the

sequel. Since now, however, it may be noted that EBG is intrinsically a metalevel process (acting on an object–level program), and that metastatements are involved since the beginning, like the declaration of operationality, which is a property of predicates, not of terms.

Variables starting with $ are metavariables, that can be instantiated to *names* of object–level expressions.

```
ebg_rp([$A],[$GenA],[$GenA]):-
        theory_fact($A).

ebg_rp([$A],[$GenA],[$GenA]):-
        operational($A),!,
        solve($A).

ebg_rp([$A|$B],[$GenA|$GenB],$G):-
        different($B,[]),
        ebg_rp([$A],[$GenA],$GA),
        ebg_rp($B,$GenB,$GB),
        append($GA,$GB,$G).

ebg_rp([$A],[$GenA],$G):-
        theory_clause($GenA,$GenB),
        different($GenB,[]),
        copy([$GenA|$GenB],[$A|$B]),
        ebg_rp($B,$GenB,$G).

copy([$GenA|$GenB],[$A,$B]):-
        copy_term($GenB,$NewGenB),
        append([$A],$NewGenB,$List),
        copy_term([$GenA|$GenB],$List),
        copy_tail($List,$B).

copy_tail([$T|$C],$C).
```

An example can better explain the functioning of the algorithm. The following, so–called suicide example is due to [DM86]. The Domain Theory is as follows:

```
kill(A,B):-hate(A,B),possess(A,C),weapon(C).
hate(W,W):-depressed(W).
possess(U,V):-buy(U,V).
weapon(Z):-gun(Z).
```

The Training Example is a set of facts:

```
depressed(john).
buy(john,obj1).
gun(obj1).
```

The Operational predicates are defined by means of the following facts:

```
operational(<depressed>).
operational(<buy>).
operational(<gun>).
```

where the constants in angle brackets are, in the Reflective Prolog syntax, names of predicates.
The Target Concept is the binary predicate kill. Given the query:

```
:-ebg_rp([<kill>("john","john")],[<kill>($X,$Y)],$Result).
```

the variabile Result is instantiated to the list:

```
[<depressed>($X),<buy>($X,$Y),<gun>($Y)]
```

so that we get the clause:

```
kill(X,X):-depressed(X),buy(X,Y),gun(Y).
```

which represents a generalization of the Training Example, usable for further cases.

3. Theory reformulation

Once the output of EBG has been obtained, the issue becomes whether to consider it together with or separated from the other clauses. The simple addition would give rise to the known "lemma generation problem" [Sou], i.e. it would be redundant and would cause unwanted repeated answers to new queries.

To solve this problem, we will take the approach, not present in the literature, of building a new domain theory DT' including the learned definitions without redundancy. In so doing, two aspects will be relevant. On the one hand, DT' and DT will not be equivalent under clause subsumption, since the objective of learning is precisely to get a DT' that is a shorter, ready-made way of defining the concept predicate thus encompassing the learning example. On the other hand, all and only the consequences of DT have to be consequences of DT'. With the usual declarative semantics of Horn clauses [Ll87], this last aspect amounts to require the equivalence of DT and DT' as equality of their least Herbrand models.

We will construct the theory DT' by means of a set of definitions. Being DT' a set resulting from the application of EBG, we will call it Resulting Set.

Definition 1:
The Resulting Set (RS) is the set of clauses obtained from the application of EBG with the given Domain Theory, a Training Example and a fixed Operationality Criterion.

Definition 2:
A clause A:−B1,..Bn is said to be below the operationality level if one of the following conditions holds:

- *operational(A)∈ OC.*
- *there exists a sequence of goals G1,..Gh such that:*
  1) *G1=Xθ where operational(X)∈ OC and θ is a substitution.*
  2) *Gh is of the form :−K1,..Kr and ∃ Kj which unifies with A.*

Definition 3:
We call Completion Theory (CT) the set of all the clauses of DT that are below the operationality level.

Definition 4:
A query Q is said to be above the operationality level if there exists a sequence Q=G1,..Gh of goals such that Gh has the form: ?−K1,..Kr where each Ki (1≤i≤r) unifies with the head of a clause that is below the operationality level and there exists at least one Kj such that Kj=Xθ with operational(X)∈ OC, where θ is a substitution.

Definition 5:
The Multiple Resulting Set (MRS) is the set of clauses obtained from the application of EBG with the given Domain Theory, a Training Example, a fixed Operationality Criterion and, as input, the set of queries which are above the operationality level.

Definition 6:
We define Completed Resulting Set (CRS) the set resulting from the union of the Multiple Resulting Set and the Completion Theory. CRS=MRS∪CT.

For instance, considering the previous example we have the following sets:

```
MRS:
kill(X,X):-depressed(X),buy(X,C),gun(C).
hate(X,X):-depressed(X).
possess(X,C):-buy(X,C).
weapon(C):-gun(C).
```

CT is empty because all the operational predicates are facts. The (Completed) Resulting Set is thus MRS∪CT=MRS.

Theorem 1:
The least Herbrand models of the two sets DTUTE *and* CRSUTE *are equal.*

The proof is in [Be93].

It is not possible to get the strongest equivalence of the two theories [Ma86]. Maps T_P [Ll87] are not equivalent for the two programs CRSUTE and DTUTE, as can be easily seen. Thus the subsumption equivalence does not hold, since the equality of maps T_P is a prerequisite for that kind of equivalence.

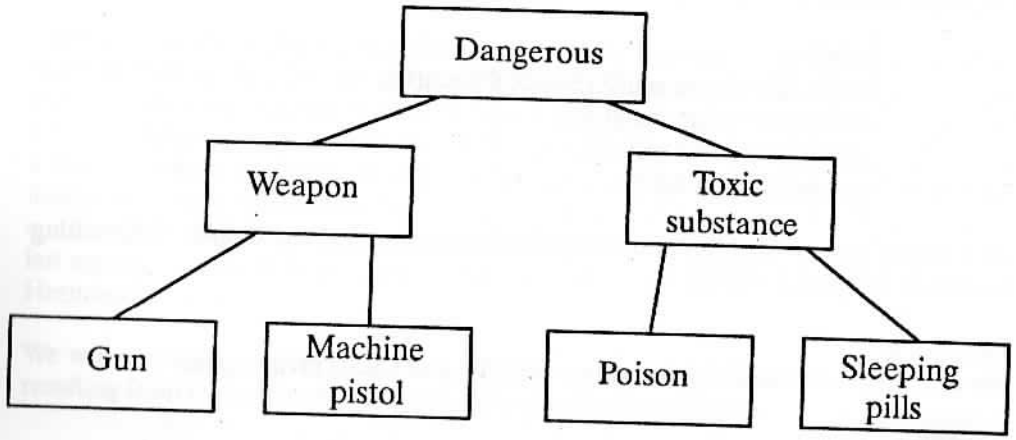
4. Theory revision

When a predicate appearing in conditions of rules is not defined by further rules, the EBG process in the previous 'standard' form (theory reformulation) is of no assistance. In a sense, the knowledge represented by rules is incomplete, leaving the interpretation of some predicates completely open.

The training example may again be taken as guiding the interpretation, but this time provided that it is used together with some additional knowledge. One useful heuristics is to abstract single instances provided by the training example to higher classes of an abstraction hierarchy.

In the above example, 'gun' is an instance of the class 'weapon', abstracting to which also e.g. 'machine−pistol' is accommodated. 'sleeping pills' do not belong to the class 'weapon' but may be used for suicide; a superclass of both may be expressed by the predicate 'dangerous', and so on.
In the following, two extensions of the standard EBG algorithm are shown, aimed at coping with such situations: the first one treats abstraction hierarchies with constraints, the second one deals with exceptions and negative cases (they are shown separately, but can be combined).

Abstraction hierarchies with constraints
If we consider again the previous example concerning suicide we can see how to obtain a more abstract clause using the following taxonomy:



which is expressed as follows:

```
weapon(<gun>).
weapon(<machine_pistol>).
dangerous(<weapon>).
toxic_substance(<poison>).
toxic_substance(<sleeping_pills>).
dangerous(<toxic_substance>).
```

The predicate 'compatible' expresses the fact that a class is compatible with its siblings and therefore with its superclass as a whole. Compatibility is not a characteristic peculiar of the hierarchy, i.e. it is not static, but depends on the particular example being dealt with. For instance the fact:

(*) compatible(<weapon>,<dangerous>).

states that the class weapon is compatible with its superclass dangerous, thus the abstraction is allowed in this particular TE.

The predicate 'belongs_to' specifies the starting point of the abstraction. For instance, in the example the user could specify:

```
belongs_to(<gun>,<weapon>).
```

A new algorithm is obtained dealing with taxonomies or hierarchy trees, by placing on top of the algorithm shown before the following clause (where metavariables starting with # denote names of predicates like <weapon>, <gun>, etc.)

```
ebg_rp([$A],[#Pred($Args)],$Chain):−
                belongs_to(#Pred,#Class),
                backtracks(#Class,#Pred,$Branch),
                append($Branch,#Item($Args),$Chain),
                !,solve($A).
```

The auxiliary predicate 'backtracks' allows climbing the hierarchy up to the most abstract node compatible with the constraints, given by means of the binary predicate 'compatible'.

```
backtracks(#Class,#Subclass,$ListA):−
                solve(#Superclass(#Class)),
                compatible(#Superclass,#Class),
                backtracks(#Superclass,#Class,$ListB),
                append($ListB,#Superclass(#Class)),$ListA).
backtracks(#Class,#Subclass,[#Class(#Subclass)]).
```

In the example, the extended EBG generates the final clause:

```
solve(<kill>($X,$X)):-
        solve(<depressed>($X)),
        solve(<buy>($X,$Y)),
        solve(<dangerous>("#P")),
        solve(#P("#Q")),
        solve(#Q($Y)).
```

which can be used to prove suicide with all the dangerous substances previously shown and not only with the gun of the Training Example.
On the contrary, the lack of a fact 'compatible' means that the abstraction is not possible (for the purpose represented by the given Training Example). For instance, if (*) is not given, the generated clause is simply:

```
solve(<kill>($X,$X)):-
        solve(<depressed>($X)),
        solve(<buy>($X,$Y)),
        solve(<weapon>("#P")),
        solve(#P($Y)).
```

and does not generalize up to dangerous, but only to weapon.

Predicates 'belongs_to' and 'compatible' are then domain-dependent directives for the abstraction process. They must be suitably specified by the user with respect to the particular TE, in order to ensure coherence of the abstraction process.
If we consider the theory presented in [EK89], we can see an analogy between the set A of abducible predicates, from which the abduction process can start, and the set of first arguments of the directives 'belongs_to'.

Exceptions and negative cases
When dealing with abstractions there is the problem of considering possible exceptions. We will cope with this problem, with no loss of generality, by expressing that two sets of items of a class share a certain property or, on the contrary, that they don't share it. We use facts of the form:

```
equival(#Property,#Class,$ListA,$ListB).
exclude(#Property,#Class,$ListA,$ListB).
```

where $ListA and $ListB represent the two sets of items.

As an example, consider the case of vehicles in a park. A class vehicle may include, among others, the items car, truck and fire_truck, as represented by the following facts:

```
vehicle(<car>).
vehicle(<truck>).
```



```
vehicle(<fire_truck>).
```

The Domain Theory states that if a vehicle is in a park it will get a fine:

```
get_fine(X):-cant_go(X,Y).
cant_go(X,Y):-vehicle(X),park(Y).
vehicle(X):-car(X).
```

The Training Example gives an instance of park and car, and states that car is an item of the class vehicles, which is divided in two subclasses of items sharing the characteristic of not being or being allowed to go in the park, respectively:

```
park(central_park).
car(car1).

belongs_to(<car>,<vehicle>).

exclude(<cant_go>,<vehicle>,[car,truck],[fire_truck]).
```

The clause thus generated by the algorithm is:

```
solve(<get_fine>($X)):-
        solve(<vehicle>("#P")),
        solve(<neutral>(#P($X))),
        solve(<park>($Y)).
```

dealing with the exceptions, i.e. stating that a fire truck would not get a fine since it does not share with the other items of the class vehicle the property of not being allowed to go in the park.
With the algorithm presented previously, which deals with the abstractions without exceptions, it is not possible to give this distinction and therefore the clause generated would have been:

```
solve(<get_fine>($X)):-
        solve(<vehicle>(#P)),
        solve(#P($X))),
        solve(<park>($Y)).
```

stating, incorrectly, that every vehicle in a park is subject to fines (fire trucks included).

The original algorithm is changed by adding the following clauses:

```
ebg_rp([$A],[#Pred($Args)],[#Pred($Args)]):-
        exclude(#Pred,#Class,$ListA,$ListB),
        solve_not(<divided>(#Class)),
```

```
                divide(#Class,#Pred,$Args,$ListA,$ListB),
                theory_add(<divided>(#Class)).

ebg_rp([$A],[#Pred($Args)],[#Pred($Args)]):-
                equival(#Pred,#Class,$ListA,$ListB),
                solve_not(<divided>(#Class)),
                divide(#Class,#Pred,$Args,$ListA,$ListB),
                theory_add(<divided>(#Class)).
```

where the solve_not is a Reflective Prolog metapredicate allowing negative conclusions to be derived,
and by modifying the clause dealing with abstraction in:

```
ebg_rp([$A],[#Pred($Args)],[#Class(#Item),<neutral>(#Item($Args))]):-
        belongs_to(#Pred,#Class),!,solve($A).
```

In order the generated clauses to properly function, the following clauses have to be also added to the program:

```
solve(<neutral>(#P($X))):-verify_pos(#P),solve(#P($X)).
solve(<neutral>(#P($X))):-verify_neg(#P),solve_not(#P($X)).
verify_pos(#P):-pos([$List]),item_of(#P,[$List]).
verify_neg(#P):-neg([$List]),item_of(#P,[$List]).
```

The division algorithm (definition of predicate decide) is presented in appendix A.

Two remarks are in order at this point.
First, the additional knowledge, represented by both taxonomies and partitions of items into two lists of positive and negative instances of a class with respect to a given property, is not intended to complement the Domain Theory, i.e. is not considered to have the same strength of the set of stipulating rules the Domain Theory consists of. Rather, it is viewed as pragmatic knowledge provided by cases and thus containing relevant but fragmentary information more than classifications of wide scope and general value.
Second, the examples were presented as if the taxonomies and the exceptions were all known in advance. The extended algorithm, however, is such that they can be dynamically arranged as long as new cases (Training Examples) are added to previous ones, thus building hierarchies and lists of positive and negative items incrementally.

In the following we give the definitions to construct the (Completed) Resulting Set in both cases (abstraction and exceptions). Definition 4 is reformulated into definition 7 to deal with the abstraction, with which the generated clauses contain predicates representing classes of the taxonomy.

Definition 7:
A query Q is said to be above the operationality level if there exists a sequence $Q=G1,..Gh$ of goals such that Gh has the form: $?-K1,..Kr$ where each Ki ($1\leq i\leq r$) unifies with the head of a clause that is below the operationality level and there exists at least one Kj such that $Kj=X\theta$ where $T:-C1,..X,..Cn$ is the clause generated by EBG, where θ is a substitution.

Definition 8:
We call Class Set (CS) the set of classes used for the abstration. An item $X \in CS$ has the form:
Class(Item).

Definition 9:
We call Positive Set (PS) of a class C the set $PS=\{Y: pos(X)$ with Y item of the list $X\}$. In a similar way we define the Negative Set (NS).
We call Gender Set the union of NS and PS. $GS= PS \cup NS$.

It is possible now to construct the MRS using definition 5. The CRS is the union of MRS, CT and CS for the case of simple abstraction. When dealing with exceptions the CRS is $MRS \cup CT \cup CS \cup GS$.

The cases presented in this section deal with abstraction supported by a taxonomy, which represents knowledge external to the Domain Theory.
This knowledge is integrated by the algorithm in the clauses that it generates. Thus the new set CRS is wider in terms of the number of queries that can be proved using it. Formally speaking we can state that:

The least Herbrand model of $CRS \cup TE$ contains the least Herbrand model of $DT \cup TE$.

To put it in another way, the set of consequences of CRSUTE is equal to the set of consequences of DTUTEU the set of the facts expressing the taxonomy and the classification of positive and negative items of a class.

5. Concluding remarks

This paper is intended as a first step towards a complete formalization of EBG, and towards making theory revision in a logic programming context.
A main point we have shown is that domain independent abstraction algorithms can be defined, that are easily understandable by the user, which can guide the process by means of simple directives.
We believe in the importance of a suitable metaprogramming framework, where different levels of knowledge can be clearly distinguished, and where tools are available for defining in a uniform language abstraction algorithms, directives and constraints.
In such a framework, the user could even interact with the system, and define her own specific abstraction methodologies (instead of simple directives).
In Reflective Prolog, the tools are *naming* and *reflection* (predicates solve and solve_not).

Appendix A

Here is the division algorithm that divides a class in subclasses containing items sharing the same property. The algorithm iteratively examines facts 'exclude' and 'equival' and constructs two lists containig respectively positive items and negative items (i.e. items that satisfy that property and items that do not).

```
divide(#Class,<not>,#Pred($Args),$ListA,$ListB):-
        item_of(#Pred,$ListA),
        negative([],$ListA).

divide(#Class,<not>,#Pred($Args),$ListA,$ListB):-
        item_of(#Pred,$ListB).
        negative([],$ListB).

divide(#Class,#Pred,$Args,$ListA,$ListB):-
        item_of(#Pred,$ListA),
        positive($ListA,[]).

divide(#Class,#Pred,$Args,$ListA,$ListB):-
        item_of(#Pred,$ListB),
        positive($ListB,[]).

positive($Poslist,$Neglist):-
        has_item_of($ListA,$Poslist),
        equival(#Pred,#Class,$ListA,$ListB),
        solve_not(<used>("equival(#Class,$ListA,$ListB)")),
        merge($ListA,$Poslist,$Temposlist),
        merge($ListB,$Temposlist,$Newposlist),
        theory_add(<used>("equival(#Class,$ListA,$ListB")),
        positive($Newposlist,$Neglist).


positive($Poslist,$Neglist):-
        has_item_of($ListA,$Poslist),
        exclude(#Pred,#Class,$ListA,$ListB),
        solve_not(<used>("exclude(#Class,$ListA,$ListB)")),
        merge($ListA,$Poslist,$Newposlist),
        merge($ListB,$Neglist,$Newneglist),
        theory_add(<used>("exclude(#Class,$ListA,$ListB)")),
        negative($Newposlist,$Newneglist).

positive($Poslist,$Neglist):-
        theory_add(<pos>("$Poslist")),
        theory_add(<neg>("$Neglist")).
```



```
negative($Poslist,$Neglist):-
        has_item_of($ListA,$Neglist),
        equival(#Pred,#Class,$ListA,$ListB),
        solve_not(<used>("equival(#Class,$ListA,$ListB)")),
        merge($ListA,$Neglist,$Temposlist),
        merge($ListB,$Temposlist,$Newneglist),
        theory_add(<used>("equival(#Class,$ListA,$ListB)")),
        negative($Poslist,$Newneglist).

negative($Poslist,$Neglist):-
        has_item_of($ListA,$Neglist),
        exclude(#Pred,#Class,$ListA,$ListB),
        solve_not(<used>("exclude(#Class,$ListA,$ListB)")),
        merge($ListB,$Poslist,$Newposlist),
        merge($ListA,$Neglist,$Newneglist),
        theory_add(<used>("exclude(#Class,$ListA,$ListB)")),
        positive($Newposlist,$Newneglist).

negative($Poslist,$Neglist):-
        theory_add(<pos>("$Poslist")),
        theory_add(<neg>("$Neglist")).
```

Predicates 'exclude' and 'equival' are symmetric with respect to their last two arguments, clauses that deal with symmetry are therefore necessary:

```
symmetric(<exclude>).
symmetric(<equival>).
solve(#P(#Property,#Class,$X,$Y)):-
        symmetric(#P),solve(#P(#Property,#Class,$Y,$X)).
```

Appendix B

Reflective Prolog is a logic programming language aimed at expressing and using knowledge and metaknowledge in a uniform way. It replaces the metalogic features of Prolog, which consist of a set of separate, pre−defined predicates, are mainly implementation−based and lack a formal semantics, with an organic framework where a metalanguage is precisely defined for the object−language of Horn clauses, and the two languages are integrated by means of reflection rules. This integration is formally given both a procedural semantics, as an extension of the SLD resolution, and a declarative semantics, as an extension of the least Herbrand model semantics.
The metalanguage relies on metaterms, which act as names for the object−level terms and predicates, and metavariables, ranging over metaterms.
For instance, "c", "V", "f", <p> are metaterms acting as names for the object−level constant symbol c, variable symbol V, function symbol f and predicate symbol p

respectively. Thus, in a compositional way, "f"("c","V") is the metaterm representing the object−level term f(c,V) and <p>("c","V") is the metaterm representing the object−level predicate p(c,V). Function metavariables (written with first character $) range over metaterms in general, and predicate metavariables (written with first character #) range over names of predicates.
Therefore, knowledge can be structured at different levels. For instance, the treatment of symmetric relations is obtained as follows.
An object−level fact concerning the relation friend, and a metalevel fact stating that friend is a symmetric relation, are represented by:

```
friend(a,b).
symmetric(<friend>).
```

Additional inference rules can be defined declaratively, by using the distinguished predicates solve and solve_not, which take as argument a metaterm representing a predicate. For instance:

```
solve(#P($X,$Y)):−symmetric(#P),solve(#P($Y,$X)).
```

states that if the relation symbol denoted by the predicate metavariable #P has been asserted as symmetric, then the atom in the conclusion denoted by #P($X,$Y) follows from the atom in the condition denoted by #P($Y,$X).
On the contrary, given

```
father(a,b).
non_symmetric(<father>).
solve_not(#P($X,$Y)):−non_symmetric(#P),solve(#P($Y,$X)).
```

if #P($Y,$X) is true (is derived) then #P($X,$Y) is false (cannot be derived).
Derivation by resolution is extended with reflection, which "reflects up" an object−level goal to the additional inference (solve and solve_not) rules, and vice versa for reflection down. Reflection implies transformation of predicates and terms into metaterms (up) or vice versa (down). In the first example, on failure of the query

```
?−friend(b,a).
```

at the object level, a reflection up takes place, generating the goal <friend>("b","a"), that matches with the head of the solve rule with the unifier {#P/<friend>, $X/"b", $Y/"a"}.
Since symmetric(<friend>) succeeds, the next subgoal is solve(<friend>("a","b")) which by reflection down reduces to friend(a,b) which succeeds at the object level.

References

[Be93] S. Bertarello, "Estensione delle tecniche di Explanation−Based Learning mediante l'uso dell'abduzione", Degree Thesis in Computer Science (G.A. Lanzarone, S. Costantini supervisors), University of Milano, 1993

[CL93] S. Costantini, G.A. Lanzarone, "A Metalogic Programming Approach: Language, Semantics and Applications", Journal of Experimental and Theoretical Artificial Intelligence, 1, 1993

[DM86] G. DeJong, R. Mooney, "Explanation−Based Learning: An Alternative View", Machine Learning, 1, 1986, 145−176

[DK90] B. Duval, Y. Kodratoff, "A Tool for the Management of Incomplete Theories: Reasoning about Explanation", Machine Learning, an Artificial Intelligence Approach, 1990

[El89] T. Ellman, "Explanation−Based Learning: a Survey of Programs and Perspectives", ACM Computing Surveys, vol. 21, n.2, June 1989, 163−221

[EK89] K. Eshghi, R. Kowalski, "Abduction Compared with Negation by Failure", Procs. 6th Int. Conf. on Logic Programming, MIT Press, 1989, 234−254

[KM87] S.T. Kedar−Cabelli, L.T. McCarty, "Explanation−Based Generalization as Resolution Theorem Proving", Procs. 4th Int. Workshop on Machine Learning, 1987

[Ll87] J.W. Lloyd, "Foundations of Logic Programming", 2nd edition, Springer−Verlag, New York, 1987

[Ma86] M.J. Maher, "Equivalence of Logic Programs", in: third Int. Conf. on Logic Programming, 1986

[Sou] R.W. Southwick, "The Lemma Generation Problem", Imperial College Technical Report, London

[VB88] A.F. van Harmelen, A. Bundy, "Explanation−Based Generalization=Partial Evaluation", Artificial Intelligence, an International Journal, vol. 36, issue 36, 1988, 401−412